Docket No. 042390.P8098X
Express Mail No.: EL466331454US

UNITED STATES PATENT APPLICATION

FOR

MANAGING A SECURE PLATFORM USING A HIERARCHICAL EXECUTIVE ARCHITECTURE IN ISOLATED EXECUTION MODE

INVENTORS:

CARL M. ELLISON

ROGER A. GOLLIVER

HOWARD C. HERBERT

DERRICK C. LIN

FRANCIS X. MCKEEN

GIL NEIGER

KEN RENERIS

JAMES A. SUTTON

SHREEKANT S. THAKKAR

MILLIND MITTAL

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Blvd., 7th Floor
Los Angeles, CA 90025-1026
(714) 557-3800

CROSS-REFERENCES TO RELATED APPLICATIONS

This is a continuation-in-part of U.S. Patent Application No. 09/539,344 filed March 31, 2000.

BACKGROUND

1. Field of the Invention

This invention relates to microprocessors. In particular, the invention relates to processor security.

2. Description of Related Art

Advances in microprocessor and communication technologies have opened up many opportunities for applications that go beyond the traditional ways of doing business. Electronic commerce (E-commerce) and business-to-business (B2B) transactions are now becoming popular, reaching the global markets at a fast rate. Unfortunately, while modern microprocessor systems provide users convenient and efficient methods of doing business, communicating and transacting, they are also vulnerable for unscrupulous attacks. Examples of these attacks include theft of data, virus, intrusion, security breach, and tampering, to name a few. Computer security, therefore, is becoming more and more important to protect the integrity of the computer systems and increase the trust of users.

Threats caused by unscrupulous attacks may be in a number of forms. An invasive remote-launched attack by hackers may disrupt the normal operation of a system connected to thousands or even millions of users. A virus program may corrupt code and/or data of a single-user platform.

Existing techniques to protect against attacks have a number of drawbacks. Anti-virus programs can only scan and detect known viruses. Security co-processors or smart cards using cryptographic or other security techniques have limitations in speed performance, memory capacity, and flexibility. Redesigning operating systems creates software compatibility issues and causes tremendous investment in development efforts.

BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

Figure 1A is a diagram illustrating a logical architecture according to one embodiment of the invention.

Figure 1B is a diagram illustrating accessibility of various elements in the operating system and the processor according to one embodiment of the invention.

Figure 1C is a diagram illustrating a computer system in which one embodiment of the invention can be practiced.

Figure 2 is a diagram illustrating an executive subsystem according to one embodiment of the invention.

Figure 3 is a diagram illustrating a processor executive handler shown in Figure 2 according to one embodiment of the invention.

Figure 4 is a diagram illustrating a processor executive shown in Figure 2 according to one embodiment of the invention.

Figure 5 is a diagram illustrating an operating system executive shown in Figure 2 according to one embodiment of the invention.

Figure 6 is a diagram illustrating a boot-up code shown in Figure 2 according to one embodiment of the invention.

Figure 7 is a flowchart illustrating a process to manage a secure platform according to one embodiment of the invention.

Figure 8 is a flowchart illustrating a process to boot up platform according to one embodiment of the invention.

Figure 9 is a flowchart illustrating a process to execute an isolated create instruction according to one embodiment of the invention.

Figure 10 is a flowchart illustrating a process to handle a processor executive according to one embodiment of the invention.

Figure 11 is a flowchart illustrating a process to handle an operating system executive according to one embodiment of the invention.

DESCRIPTION

The present invention is a method and apparatus to manage a secure platform. A processor executive (PE) handles an operating system executive (OSE) in a secure environment. The secure environment has a platform key (PK) and is associated with an isolated memory area in the platform. The OSE manages a subset of an operating system (OS) running on the platform. The platform has a processor operating in one of a normal execution mode and an isolated execution mode. The isolated memory area is accessible to the processor in the isolated execution mode. A PE supplement supplements the PE with a PE manifest representing the PE and a PE identifier to identify the PE. A PE handler handles the PE using the PK and the PE supplement.

A boot-up code boots up the platform following a power on. The secure environment includes an OSE supplement to supplement the OSE with an OSE manifest representing the OSE and an OSE identifier to identify the OSE. The PE handler includes a PE loader, a PE manifest verifier, a PE verifier, a PE key generator, a PE identifier logger, and a PE entrance/exit handler. The PE loader loads the PE and the PE supplement from a PE memory into the isolated memory area using a parameter block provided by the boot-up code. The PE manifest verifier verifies the PE manifest. The PE verifier verifies the PE using the PE manifest and a constant derived from the PK. The PE key generator generates a PE key using the PK. The PE key generator includes a PE key combiner to combine the PE identifier and the PK. The combined PE identifier and the PK correspond to the PE key. The PE identifier logger logs the PE identifier in a storage. The PE entrance/exit handler handles a PE entry and a PE exit.

The OSE handler includes an OSE loader, an OSE manifest verifier, an OSE verifier, an OSE key generator, an OSE identifier logger, and an OSE entrance/exit handler. The OSE loader loads the OSE and the OSE supplement into the isolated memory area. The OSE manifest verifier verifies the OSE manifest. The OSE verifier verifies the OSE. The OSE key generator generates an OSE key. The OSE identifier logger logs the OSE identifier in a storage. The OSE entrance/exit handler handles an OSE entry and an OSE exit. The OSE key generator includes a binding key generator and an OSE key combiner. The binding key generator generates a binding key (BK) using the PE key. The OSE key combiner combines the OSE identifier and the BK. The combined OSE identifier and the BK correspond to the OSE key.

The OSE includes a module loader and evictor, a key binder and unbinder, a page manager, an interface handler, a scheduler and balancer, and an interrupt handler.

The module loader and evictor loads and evicts a module into and out of the isolated memory area, respectively. The module is one of an application module, an applet module, and a support module. The page manager manages paging in the isolated memory area. The interface handler handles interface with the OS. The key binder and unbinder includes an applet key generator to generate an applet key associating with the applet module. The applet key generator includes an applet key combiner to combine the OSE key with an applet identifier identifying the applet module. The combined OSE key and the applet identifier correspond to the applet key.

The boot up code includes a PE locator, a PE recorder, and an instruction invoker. The PE locator locates the PE and the PE supplement. The PE locator transfers the PE and the PE supplement into the PE memory at a PE address. The PE recorder records the PE address in the parameter block. The instruction invoker executes an isolated create instruction which loads the PE handler into the isolated memory area. The isolated create instruction performs an atomic non-interruptible sequence. The atomic sequence includes a number of operations: a physical memory operation, an atomic read-and-increment operation, an isolated memory area control operation, a processor isolated execution operation, an PE handler loading operation, a PE handler verification, and an exit operation. The physical memory operation verifies if the processor is in a flat physical page mode. The atomic read-and-increment operation reads and increments a thread count register in a chipset. The read-and-increment operation determines if the processor is the first processor in the isolated execution mode. The isolated memory area control operation configures the chipset using a configuration storage. The processor isolated execution operation configures the processor in the isolated execution mode. The processor isolated execution operation includes a chipset read operation and a processor configuration operation. The chipset read operation reads the configuration storage in the chipset when the processor is not a first processor in the isolated execution mode. The processor configuration operation configures the processor according to the configuration storage when the processor is not a first processor in the isolated execution mode. The PE handler loading operation loads the PE handler into the isolated memory area. The PE handler verification verifies the loaded PE handler. The exit operation transfers control to the loaded PE handler.

The chipset includes at least one of a memory controller hub (MCH) and an input/output controller hub (ICH). The storage is in an input/output controller hub (ICH) external to the processor.

In the following description, for purposes of explanation, numerous details are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that these specific details are not required in order to practice the present invention. In other instances, well-known electrical structures and circuits are shown in block diagram form in order not to obscure the present invention.

ARCHITECTURE OVERVIEW

One principle for providing security in a computer system or platform is the concept of an isolated execution architecture. The isolated execution architecture includes logical and physical definitions of hardware and software components that interact directly or indirectly with an operating system of the computer system or platform. An operating system and the processor may have several levels of hierarchy, referred to as rings, corresponding to various operational modes. A ring is a logical division of hardware and software components that are designed to perform dedicated tasks within the operating system. The division is typically based on the degree or level of privilege, namely, the ability to make changes to the platform. For example, a ring-0 is the innermost ring, being at the highest level of the hierarchy. Ring-0 encompasses the most critical, privileged components. In addition, modules in Ring-0 can also access to lesser privileged data, but not vice versa. Ring-3 is the outermost ring, being at the lowest level of the hierarchy. Ring-3 typically encompasses users or applications level and executes the least trusted code. It is noted that the level of the ring hierarchy is independent to the level of the security protection of that ring.

Figure 1A is a diagram illustrating a logical operating architecture 50 according to one embodiment of the invention. The logical operating architecture 50 is an abstraction of the components of an operating system and the processor. The logical operating architecture 50 includes ring-0 10, ring-1 20, ring-2 30, ring-3 40, and a processor nub loader 52. The processor nub loader 52 is an instance of a processor executive (PE) handler. The PE handler is used to handle and/or manage a processor executive (PE) as will be discussed later. The logical operating architecture 50 has two modes of operation: normal execution mode and isolated execution mode. Each ring in

the logical operating architecture 50 can operate in both modes. The processor nub loader 52 operates only in the isolated execution mode.

Ring-0 10 includes two portions: a normal execution Ring-0 11 and an isolated execution Ring-0 15. The normal execution Ring-0 11 includes software modules that are critical for the operating system, usually referred to as kernel. These software modules include primary operating system (e.g., kernel) 12, software drivers 13, and hardware drivers 14. The isolated execution Ring-0 15 includes an operating system (OS) nub 16 and a processor nub 18. The OS nub 16 and the processor nub 18 are instances of an OS executive (OSE) and processor executive (PE), respectively. The OSE and the PE are part of executive entities that operate in a secure environment associated with the isolated area 70 and the isolated execution mode. The processor nub loader 52 is a protected bootstrap loader code held within a chipset in the system and is responsible for loading the processor nub 18 from the processor or chipset into an isolated area as will be explained later.

Similarly, ring-1 20, ring-2 30, and ring-3 40 include normal execution ring-1 21, ring-2 31, ring-3 41, and isolated execution ring-1 25, ring-2 35, and ring-3 45, respectively. In particular, normal execution ring-3 includes N applications 42_1 to 42_N and isolated execution ring-3 includes K applets 46_1 to 46_K.

One concept of the isolated execution architecture is the creation of an isolated region in the system memory, referred to as an isolated area, which is protected by both the processor and chipset in the computer system. Portions of the isolated region may also be in cache memory. Access to this isolated region is permitted only from a front side bus (FSB) of the processor, using special bus (e.g., memory read and write) cycles, referred to as isolated read and write cycles. The special bus cycles are also used for snooping. The isolated read and write cycles are issued by the processor executing in an isolated execution mode when accessing the isolated area. The isolated execution mode is initialized using a privileged instruction in the processor, combined with the processor nub loader 52. The processor nub loader 52 verifies and loads a ring-0 nub software module (e.g., processor nub 18) into the isolated area. The processor nub 18 provides hardware-related services for the isolated execution.

One task of the processor nub loader 52 and processor nub 18 is to verify and load the ring-0 OS nub 16 into the isolated area, and to generate the root of a key hierarchy unique to a combination of the platform, the processor nub 18, and the operating system nub 16. The operating system nub 16 provides links to services in the

primary OS 12 (e.g., the unprotected operating system), provides page management within the isolated area, and has the responsibility for loading ring-3 application modules 45, including applets 46_1 to 46_K, into protected pages allocated in the isolated area. The operating system nub 16 may also load ring-0 supporting modules.

The operating system nub 16 may choose to support paging of data between the isolated area and ordinary (e.g., non-isolated) memory. If so, then the operating system nub 16 is also responsible for encrypting and hashing the isolated area pages before evicting the page to the ordinary memory, and for checking the page contents upon restoration of the page. The isolated mode applets 46_1 to 46_K and their data are tamper-resistant and monitor-resistant from all software attacks from other applets, as well as from non-isolated-space applications (e.g., 42_1 to 42_N), drivers and even the primary operating system 12. The only software that can interfere with or monitor the applet's execution is the processor nub loader 52, processor nub 18 or the operating system nub 16.

Figure 1B is a diagram illustrating accessibility of various elements in the operating system 10 and the processor according to one embodiment of the invention. For illustration purposes, only elements of ring-0 10 and ring-3 40 are shown. The various elements in the logical operating architecture 50 access an accessible physical memory 60 according to their ring hierarchy and the execution mode.

The accessible physical memory 60 includes an isolated area 70 and a non-isolated area 80. The isolated area 70 includes applet pages 72 and nub pages 74. The non-isolated area 80 includes application pages 82 and operating system pages 84. The isolated area 70 is accessible only to elements of the operating system and processor operating in isolated execution mode. The non-isolated area 80 is accessible to all elements of the ring-0 operating system and to the processor.

The normal execution ring-0 11 including the primary OS 12, the software drivers 13, and the hardware drivers 14, can access both the OS pages 84 and the application pages 82. The normal execution ring-3, including applications 42_1 to 42_N, can access only to the application pages 82. Generally applications can only access to their own pages, however, the OS typically provides services for sharing memory in controlled methods. Both the normal execution ring-0 11 and ring-3 41, however, cannot access the isolated area 70.

The isolated execution ring-0 15, including the OS nub 16 and the processor nub 18, can access to both of the isolated area 70, including the applet pages 72 and the

nub pages 74, and the non-isolated area 80, including the application pages 82 and the OS pages 84. The isolated execution ring-3 45, including applets 46_1 to 46_K, can access only applet pages 72. The applets 46_1 to 46_K reside in the isolated area 70. In general, applets can only access their own pages; however, the OS nub 16 can also provides services for the applet to share memory (e.g., share memory with other applets or with non-isolated area applications).

Figure 1C is a diagram illustrating a computer system 100 in which one embodiment of the invention can be practiced. The computer system 100 includes a processor 110, a host bus 120, a memory controller hub (MCH) 130, a system memory 140, an input/output controller hub (ICH) 150, a non-volatile memory, or system flash, 160, a mass storage device 170, input/output devices 175, a token bus 180, a motherboard (MB) token 182, a reader 184, and a token 186. The MCH 130 may be integrated into a chipset that integrates multiple functionalities such as the isolated execution mode, host-to-peripheral bus interface, memory control. Similarly, the ICH 150 may also be integrated into a chipset together or separate from the MCH 130 to perform I/O functions. For clarity, not all the peripheral buses are shown. It is contemplated that the system 100 may also include peripheral buses such as Peripheral Component Interconnect (PCI), accelerated graphics port (AGP), Industry Standard Architecture (ISA) bus, and Universal Serial Bus (USB), etc. The "token bus" may be part of the USB bus, e.g., it may be hosted on the USB bus.

The processor 110 represents a central processing unit of any type of architecture, such as complex instruction set computers (CISC), reduced instruction set computers (RISC), very long instruction word (VLIW), or hybrid architecture. In one embodiment, the processor 110 is compatible with an Intel Architecture (IA) processor, such as the Pentium™ series, the IA-32™ and the IA-64™. The processor 110 includes a normal execution mode 112 and an isolated execution circuit 115. The normal execution mode 112 is the mode in which the processor 110 operates in a non-secure environment, or a normal environment without the security features provided by the isolated execution mode. The isolated execution circuit 115 provides a mechanism to allow the processor 110 to operate in an isolated execution mode. The isolated execution circuit 115 provides hardware and software support for the isolated execution mode. This support includes configuration for isolated execution, definition of an isolated area, definition (e.g., decoding and execution) of isolated instructions, generation of isolated access bus cycles, and access checking.

In one embodiment, the computer system 100 can be a single processor system, such as a desktop computer, which has only one main central processing unit, e.g. processor 110. In other embodiments, the computer system 100 can include multiple processors, e.g. processors 110, 110a, 110b, etc., as shown in Figure 1C. Thus, the computer system 100 can be a multi-processor computer system having any number of processors. For example, the multi-processor computer system 100 can operate as part of a server or workstation environment. The basic description and operation of processor 110 will be discussed in detail below. It will be appreciated by those skilled in the art that the basic description and operation of processor 110 applies to the other processors 110a and 110b, shown in Figure 1C, as well as any number of other processors that may be utilized in the multi-processor computer system 100 according to one embodiment of the present invention.

The processor 110 may also have multiple logical processors. A logical processor, sometimes referred to as a thread, is a functional unit within a physical processor having an architectural state and physical resources allocated according to some partitioning policy. Within the context of the present invention, the terms "thread" and "logical processor" are used to mean the same thing. A multi-threaded processor is a processor having multiple threads or multiple logical processors. A multi-processor system (e.g., the system comprising the processors 110, 110a, and 110b) may have multiple multi-threaded processors.

The host bus 120 provides interface signals to allow the processor 110 or processors 110, 100a, and 110b to communicate with other processors or devices, e.g., the MCH 130. In addition to normal mode, the host bus 120 provides an isolated access bus mode with corresponding interface signals for memory read and write cycles. The isolated access bus mode is asserted on memory accesses initiated while the processor 110 is in the isolated execution mode and it is accessing memory within the isolated area. The isolated access bus mode is also asserted on instruction pre-fetch and cache write-back cycles if the address is within the isolated area address range. The isolated access bus mode is configured within the processor 110. The processor 110 responds to a snoop cycle to a cached address when the isolated access bus mode on the FSB matches the mode of the cached address.

The MCH 130 provides control and configuration of system memory 140. The MCH 130 provides interface circuits to recognize and service isolated access assertions on memory reference bus cycles, including isolated memory read and write cycles. In

addition, the MCH 130 has memory range registers (e.g., base and length registers) to represent the isolated area in the system memory 140. Once configured, the MCH 130 aborts any access to the isolated area that does not have the isolated access bus mode asserted.

The system memory 140 stores system code and data. The system memory 140 is typically implemented with dynamic random access memory (DRAM) or static random access memory (SRAM). The system memory 140 includes the accessible physical memory 60 (shown in Figure 1B). The accessible physical memory includes a loaded operating system 142, the isolated area 70 (shown in Figure 1B), and an isolated control and status space 148. The loaded operating system 142 is the portion of the operating system that is loaded into the system memory 140. The loaded OS 142 is typically loaded from a mass storage device via some boot code in a boot storage such as a boot read only memory (ROM). The isolated area 70, as shown in Figure 1B, is the memory area that is defined by the processor 110 when operating in the isolated execution mode. Access to the isolated area 70 is restricted and is enforced by the processor 110 and/or the MCH 130 or other chipset that integrates the isolated area functionalities. The isolated control and status space 148 is an input/output (I/O)-like, independent address space defined by the processor 110. The isolated control and status space 148 contains mainly the isolated execution control and status registers. The isolated control and status space 148 does not overlap any existing address space and is accessed using the isolated bus cycles. The system memory 140 may also include other programs or data that are not shown.

The ICH 150 represents a known single point in the system having the isolated execution functionality. For clarity, only one ICH 150 is shown. The system 100 may have many ICH's similar to the ICH 150. When there are multiple ICH's, a designated ICH is selected to control the isolated area configuration and status. In one embodiment, this selection is performed by an external strapping pin. As is known by one skilled in the art, other methods of selecting can be used, including using programmable configuring registers. The ICH 150 has a number of functionalities that are designed to support the isolated execution mode in addition to the traditional I/O functions. In particular, the ICH 150 includes an isolated bus cycle interface 152, the processor nub loader 52 (shown in Figure 1A), a digest memory 154, a cryptographic key storage 155, an isolated execution logical processor manager 156, and a token bus interface 159.

The isolated bus cycle interface 152 includes circuitry to interface to the isolated bus cycle signals to recognize and service isolated bus cycles, such as the isolated read and write bus cycles. The processor nub loader 52, as shown in Figure 1A, includes a processor nub loader code and its digest (e.g., cryptographic hash) value. The processor nub loader 52 is invoked by execution of an appropriate isolated instruction (e.g., Iso_Init) and is transferred to the isolated area 70. From the isolated area 80, the processor nub loader 52 copies the processor nub 18 from the system flash memory (e.g., the processor nub code 18 in non-volatile memory 160) into the isolated area 70, verifies and logs its integrity, and manages a symmetric key used to protect the processor nub's secrets. In one embodiment, the processor nub loader 52 is implemented in read only memory (ROM). For security purposes, the processor nub loader 52 is unchanging, tamper-resistant and non-substitutable. The digest memory 154, typically implemented in RAM, stores the digest (e.g., cryptographic hash) values of the loaded processor nub 18, the operating system nub 16, and any other supervisory modules (e.g., ring-0 modules) loaded into the isolated execution space. The cryptographic key storage 155 holds a symmetric encryption/decryption key that is unique for the platform of the system 100. In one embodiment, the cryptographic key storage 155 includes internal fuses that are programmed at manufacturing. Alternatively, the cryptographic key storage 155 may also be created during manufacturing with a cryptographic random number generator. The isolated execution logical processor manager 156 manages the operation of logical processors configuring their isolated execution mode support. In one embodiment, the isolated execution logical processor manager 156 includes a logical processor count register that tracks the number of logical processors participating in the isolated execution mode. The token bus interface 159 interfaces to the token bus 180. A combination of the processor nub loader digest, the processor nub digest, the operating system nub digest, and optionally additional digests, represents the overall isolated execution digest, referred to as isolated digest. The isolated digest is a fingerprint identifying the all supervisory code involved in controlling the isolated execution configuration and operation. The isolated digest is used to attest or prove the state of the current isolated execution environment.

The non-volatile memory 160 stores non-volatile information. Typically, the non-volatile memory 160 is implemented in flash memory. In one embodiment, the non-volatile memory 160 includes the processor nub 18. The processor nub 18 provides set-up and low-level management of the isolated area 70 (in the system

memory 140), including verification, loading, and logging of the operating system nub 16, and the management of the symmetric key used to protect the operating system nub's secrets. The processor nub loader 52 performs some part of the setup and manages/updates the symmetric key before the processor nub 18 and the OS nub 16 are loaded. The processor nub 18 The processor nub 18 may also provide interface abstractions to low-level security services provided by other hardware. The processor nub 18 may also be distributed by the original equipment manufacturer (OEM) or operating system vendor (OSV).

The mass storage device 170 stores archive information such as code (e.g., processor nub 18), programs, files, data, applications (e.g., applications 42_1 to 42_N), applets (e.g., applets 46_1 to 46_K) and operating systems. The mass storage device 170 may include compact disk (CD) ROM 172, floppy diskettes 174, and hard drive 176, and any other storage devices. The mass storage device 170 provides a mechanism to read machine-readable media. When implemented in software, the elements of the present invention are the code segments to perform the necessary tasks. The program or code segments can be stored in a processor readable medium or transmitted by a computer data signal embodied in a carrier wave, or a signal modulated by a carrier, over a transmission medium. The "processor readable medium" may include any medium that can store or transfer information. Examples of the processor readable medium include an electronic circuit, a semiconductor memory device, a ROM, a flash memory, an erasable programmable ROM (EPROM), a floppy diskette, a compact disk CD-ROM, an optical disk, a hard disk, a fiber optical medium, a radio frequency (RF) link, etc. The computer data signal may include any signal that can propagate over a transmission medium such as electronic network channels, optical fibers, air, electromagnetic, RF links, etc. The code segments may be downloaded via computer networks such as the Internet, an Intranet, etc.

I/O devices 175 may include any I/O devices to perform I/O functions. Examples of I/O devices 175 include a controller for input devices (e.g., keyboard, mouse, trackball, pointing device), media card (e.g., audio, video, graphics), a network card, and any other peripheral controllers.

The token bus 180 provides an interface between the ICH 150 and various tokens in the system. A token is a device that performs dedicated input/output functions with security functionalities. A token has characteristics similar to a smart card, including at least one reserved-purpose public/private key pair and the ability to

sign data with the private key. Examples of tokens connected to the token bus 180 include a motherboard token 182, a token reader 184, and other portable tokens 186 (e.g., smart card). The token bus interface 159 in the ICH 150 connects through the token bus 180 to the ICH 150 and ensures that when commanded to prove the state of the isolated execution, the corresponding token (e.g., the motherboard token 182, the token 186) signs only valid isolated digest information. For purposes of security, the token should be connected to the digest memory via the token bus 180.

A HIERARCHICAL EXECUTIVE ARCHITECTURE TO MANAGE A SECURE PLATFORM

The overall architecture discussed above provides a basic insight into a hierarchical executive architecture to manage a secure platform. The elements shown in Figures 1A, 1B, and 1C are instances of an abstract model of this hierarchical executive architecture. The implementation of this hierarchical executive architecture is a combination of hardware and software. In what follows, the processor executive, the processor executive handler, and the operating system executive are abstract models of the processor nub 18, the processor nub loader 52, and the operating system nub 16 (Figures 1A, 1B, and 1C), respectively.

Figure 2 is a diagram illustrating an executive subsystem 200 according to one embodiment of the invention. The executive subsystem 200 includes a processor executive (PE) 210, a PE supplement 220, a PE handler 230, a boot-up code 240, and a secure environment 250.

The processor executive (PE) 210 handles an operating system executive (OSE) 270 in the secure environment 250. The PE supplement 220 supplements the PE with a PE manifest 222 representing the PE and a PE identifier 224 to identify the PE. The PE handler 230 handles the PE 210 using a platform key (PK) 260 in the secure environment 250 and the PE supplement 220. The PE 210 and the PE supplement 220 are located in a PE memory 215. The PE memory 215 is located in the non-isolated memory area 80.

The PE handler 230 handles the PE 210 using the PK 260 and the PE supplement 220. The PE handler 230 obtains information to locate the PE memory 215 via a parameter block 242 provided by the boot-up code 240.

The boot-up code 240 boots up the platform following a power on. The boot-up code 240 obtains an original PE 246 and an original PE supplement 248 from a system ROM (e.g., system flash 160 as shown in Figure 1C)

The secure environment 250 includes a platform key (PK) 260, an operating system executive (OSE) 270, and an OSE supplement 280. The OSE supplement 280 supplements the OSE 270 with an OSE manifest 282 representing the OSE and an OSE identifier 284 to identify the OSE. The secure environment 250 is associated with an isolated memory area 70 (Figure 1C) in the platform. The OSE 270 manages a subset 295 of an operating system (OS) 290 running on the platform. The platform has a processor 110 operating in one of a normal execution mode 112 and an isolated execution mode 115 as shown in Figure 1C. The isolated memory area 70 is accessible to the processor 110 in the isolated execution mode 115.

Figure 3 is a diagram illustrating the PE handler 230 shown in Figure 2 according to one embodiment of the invention. The PE handler 230 includes a PE loader 310, a PE manifest verifier 320, a PE verifier 330, a PE Error Generator 340, a Constant Driver 350, a PE key generator 360, a PE identifier logger 370, and a PE entrance/exit handler 380.

The PE loader 310 loads the PE 210 and the PE supplement 220 from the PE memory 215 (Figure 2) into the isolated memory area 70 using a PE address in the parameter block 242 (Figure 2) provided by the boot-up code 240. The PE loader 310 provides a loaded PE manifest 322 and a loaded PE 312 located in the isolated memory area 70 and corresponding to the PE manifest 322 and the PE 312, respectively.

The PE manifest verifier 320 verifies the PE manifest 222 by comparing the PE manifest 222 with the loaded PE manifest 322 and generates a result to a PE error generator 340. If the verification fails, the error generator 340 generates a failure or fault condition with an error code associated with the PE manifest verification.

The PE verifier 330 verifies the PE 210 using the verified loaded PE manifest 322 and a constant 355 derived from the PK 260 by a constant deriver 350. Essentially, the PE verifier 330 compares the PE 210 with the loaded PE 312. In addition, the PE verifier 330 determines a manifest of the loaded PE 312 using the constant 355 and compares the determined PE manifest with the verified loaded PE manifest 322. The PE verifier 330 then generates a result to the PE error generator 340. If the verification fails, the error generator 340 generates a failure or fault condition with an error code associated with the PE verification.

The PE key generator 360 generates a PE key 365 using the PK 260. The PE key generator 360 includes a PE key combiner 364 to combine the PE identifier 224

and the PK 260. The combined PE identifier 224 and the PK 260 correspond to the PE key 365.

The PE identifier logger 370 logs the PE identifier 224 in a storage 375. The PE identifier logger 370 writes the PE identifier 224 into the storage 375. The storage 375 is a register located inside a chipset such as the ICH 150 shown in Figure 1C.

The PE entrance/exit handler 380 handles a PE entrance and a PE exit. The PE entrance includes obtaining the entry point in the configuration buffer of the processor 110 to represent the PE's entry handler. The PE exit returns control to the boo-up code 240.

Figure 4 is a diagram illustrating the PE 210 shown in Figure 2 according to one embodiment of the invention. The PE 210 includes an OSE loader 410, an OSE manifest verifier 420, an OSE verifier 430, an OSE Error Generator 440, an OSE key generator 460, an OSE identifier logger 470, and an OSE entrance/exit handler 480.

The OSE loader 410loads the OSE 270 and the OSE supplement 280 into the isolated memory area 70 as shown in Figure 2 using an OSE parameter block 405 provided by the OS 290. The OSE loader 410 provides a loaded OSE manifest 422 and a loaded OSE 412 located in the isolated memory area 70 and corresponding to the OSE manifest 282 and the OSE 270, respectively.

The OSE manifest verifier 420 verifies the OSE manifest 282 by comparing the OSE manifest 282 with the loaded OSE manifest 422. The OSE manifest verifier 420 generates a result to an OSE error generator 440. If the verification fails, the OSE error generator 440 generates a failure or fault condition with an error code associated with the OSE manifest verification.

The OSE verifier 430 verifies the OSE 270. Essentially, the OSE verifier 430 compares the OSE 270 with the loaded OSE 412. In addition, the OSE verifier 430 determines a manifest of the loaded OSE 412 using a root key and compares the determined OSE manifest with the verified loaded OSE manifest 422. The OSE verifier 430 then generates a result to the OSE error generator 440. If the verification fails, the OSE error generator 440 generates a failure or fault condition with an error code associated with the OSE verification.

The OSE key generator 460 generates an OSE key 465. The OSE key generator 460 includes a binding key (BK) generator 462 and an OSE key combiner 464. The binding key generator 462 generates a binding key (BK) 463 using the PE key 365 (Figure 3). The OSE key combiner 464 combines the OSE identifier 284 and the BK

463. The combined OSE identifier 284 and the BK 463 correspond to the OSE key 465.

The OSE identifier logger 470 logs the OSE identifier 284 in the storage 375. The storage 375 is a register located inside a chipset such as the ICH 150 shown in Figure 1C.

The OSE entrance/exit handler 480 handles an OSE entrance and an OSE exit. The OSE entrance initializes parameters in a frame buffer and saves appropriate control parameters and transfers control to an entrance handler. The OSE exit clears and creates appropriate return parameters and then transfers control to the exit handler,

Figure 5 is a diagram illustrating the OSE 270 shown in Figure 2 according to one embodiment of the invention. The OSE 270 includes a module loader and evictor 510, a page manager 520, an interface handler 530, a key binder and unbinder 540, a scheduler and balancer 550, and an interrupt handler 560.

The module loader and evictor 510 loads and evicts a module into and out of the isolated memory area 70, respectively. The module is one of an application module 512, an applet module 514, and a support module 516. The page manager 520 manages paging in the isolated memory area 70. The interface handler 530 handles interface with the subset 295 in the OS 290 (Figure 2). The key binder and unbinder 540 includes an applet key generator 542 to generate an applet key 545 associated with the applet module 514. The applet key generator 542 includes an applet key combiner 544 combines the OSE key 465 (Figure 4) with an applet identifier 518 identifying the applet module 514. The combined OSE key 465 and the applet identifier 518 correspond to the applet key 545.

The scheduler and balancer 550 schedules execution of the loaded modules and balances the load of the isolated execution mode. The interrupt handler 560 handles interrupts and exceptions generated in the isolated execution mode.

Figure 6 is a diagram illustrating a boot-up code shown in Figure 2 according to one embodiment of the invention. The boot up code includes a PE locator 610, a PE recorder 620, and an instruction invoker 630.

The PE locator 610 locates the original PE 246 and the original PE supplement 248. The PE locator 610 transfers the original PE 246 and the original PE supplement 248 into the PE memory 215 at a PE address 625. The PE recorder 620 records the PE address 625 in the PE parameter block 242. As discussed above, the PE handler 230

obtains the PE address 625 from the PE parameter block 242 to locate the PE 210 and the PE supplement 220 in the PE memory 215.

The instruction invoker 630 invokes and executes an isolated create instruction 632 which loads the PE handler 230 into the isolated memory area 70. The isolated create instruction 632 performs an atomic non-interruptiblle sequence 640. The atomic sequence 640 includes a number of operations: a physical memory operation 652, an atomic read-and-increment operation 654, an isolated memory area control operation 656, a processor isolated execution operation 658, an PE handler loading operation 663, a PE handler verification 664, and an exit operation 666.

The physical memory operation 652 verifies if the processor is in a flat physical page mode. The atomic read-and-increment operation 654 reads and increments a thread count register in a chipset. The read-and-increment operation 654 determines if the processor is the first processor in the isolated execution mode. The isolated memory area control operation 656 configures the chipset using a configuration storage. The processor isolated execution operation 658 configures the processor in the isolated execution mode. The processor isolated execution operation 658 includes a chipset read operation 672 and a processor configuration operation 674. The chipset read operation 672 reads the configuration storage in the chipset when the processor is not a first processor in the isolated execution mode. The processor configuration operation 674 configures the processor according to the configuration storage read by the chipset read operation 672 when the processor is not a first processor in the isolated execution mode. The PE handler loading operation 662 loads the PE handler 230 into the isolated memory area 70. The PE handler verification 664 verifies the loaded PE handler. The exit operation 666 transfers control to the loaded PE handler.

Figure 7 is a flowchart illustrating a process 700 to manage a secure platform according to one embodiment of the invention.

Upon START, the process 700 boots up the platform following power on (Block 710). The platform has a secure environment. The secure environment includes a platform key, an operating system executive (OSE), and an OSE supplement. The details of the Block 710 are shown in Figure 8. Then, the process 700 handles a processor executive (PE) using the platform key and the PE supplement (Block 720). The details of the Block 720 are shown in Figure 10. Then, the process 700 handles the OSE in the secure environment (Block 730). The details of the Block 730 are shown in Figure 11.

Next, the process 700 manages a subset of an operating system running on the platform (Block 740). The process 700 is then terminated.

Figure 8 is a flowchart illustrating the process 710 to boot up platform according to one embodiment of the invention.

Upon START, the process 710 locates the PE and the PE supplement (Block 810). Then, the process 710 transfers the PE and the PE supplement into the PE memory at a PE address (Block 820). Next, the process 710 records the PE address in a PE parameter block (Block 830). Then, the process 710 executes the isolated create instruction (Block 840). The details of the Block 840 are shown in Figure 9. The process 710 is then terminated.

Figure 9 is a flowchart illustrating the process 840 to execute an isolated create instruction according to one embodiment of the invention.

Upon START, the process 840 determines if the processor is in a flat physical page mode (Block 910). If not, the process 840 sets the processor in the flat physical page mode (Block 915) and proceeds to Block 920. Otherwise, the process 840 determines if the thread count register is zero (Block 920). This is done by reading the thread count register in the chipset to determine if the processor is the first processor in the isolated execution mode. If not, the process 840 determines that the processor is not the first processor in the system to be in the isolated execution mode. The process 840 then reads the configuration storage from the chipset (Block 925). Then, the process 840 configured the processor using the chipset configuration storage (Block 930). Then, the process 840 proceeds to Block 960.

If the thread count register is zero, the process 840 determines that the processor is the first processor in the system to be booted up with isolated execution mode. The process 840 then increments the thread count register to inform to other processors that there is already a processor being booted up in isolated execution mode (Block 935). Then, the process 840 configures the chipset and the processor in isolated execution mode by writing appropriate setting values (e.g., isolated mask and base values) in the chipset and processor configuration storage (Block 940). To configure the processor, the process 840 may also need to set up the isolated execution mode word in the control register of the processor.

Next, the process 840 loads the PE handler from the ROM internal to the chipset to the isolated memory area (Block 945). Then, the process 840 determines if the loaded PE handler is the same as the original PE handler in the ROM (Block 950). If

not, the process 840 generates a failure or fault condition with an appropriate error code (Block 955) and is then terminated. Otherwise, the process 840 transfers control to the loaded PE handler (Block 960). The process 840 is then terminated.

Figure 10 is a flowchart illustrating the process 720 to handle a processor executive according to one embodiment of the invention.

Upon START, the process 720 loads the PE and the PE supplement from a PE memory into the isolated memory area using a parameter block provided by the boot-up code (Block 1010). Next, the process 720 determines if the loaded PE manifest is the same as the original PE manifest (Block 1015). If not, the process 720 generates a failure or fault condition with appropriate error code (Block 1020) and is then terminated. Otherwise, the process 720 determines if the loaded PE has the same manifest as the loaded PE manifest (Block 1025). If not, the process 720 goes to Block 1020 and is then terminated. Otherwise, the process 720 generates a PE key using the platform key in the secure environment (Block 1030).

Then, the process 720 logs the PE identifier in a storage (Block 1035). This log storage is typically a register in an ICH. Then, the process 720 changes the entry point in the configuration buffer of the processor to prepare for an OSE entrance (Block 1040). Then, the process 720 returns to the boot-up code (Block 1045). The process 720 is then terminated.

Figure 11 is a flowchart illustrating the process 730 to handle the OSE according to one embodiment of the invention.

Upon START, the OS boots and locates the OSE and the OSE supplement in the OSE memory at an OSE address (Block 1110). Then the OS records the OSE address in an OSE parameter block (Block 1115). Next, the process 730 determines if an OSE has already been loaded (Block 1120). If yes, the process 730 is terminated. Otherwise, the process 730 loads the OSE and the OSE supplement into the isolated memory area (Block 1125).

Next, the process 730 determines if the loaded OSE manifest is the same as the original OSE manifest (Block 1130). If not, the process 730 generates a failure or fault condition with an appropriate error code (Block 1135) and is then terminated. Otherwise, the process 730 determines if the loaded OSE has the same manifest as the loaded OSE manifest (Block 1140). If not, the process 730 goes to block 1135 and is then terminated. Otherwise, the process 730 generates the OSE key using the PE key and the OSE identifier (Block 1145).

Then, the process 730 logs the OSE identifier in a storage (Block 1150). Typically, this log storage is a register in a chipset such as the ICH. Next, the process 730 clears any PE secrets or services that are not needed (Block 1155). Then, the process 730 returns to the PE's exit handler (Block 1160). The process 730 is then terminated.

While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.